Hello thank you for using this guide. This guide follows FSDG regulations. It will never ask or recommend you to install non-free software. Non-free software is dangerous and is often backdoored by the government or at the very least a pain in the butt.

This guide also assumes that you are running libreboot, canoeboot as your bios. It has been tested on Version 26.01. If you have a non-free stock bios it will not work. I am unsure of compatability with Coreboot such as the System 76 laptops. This guide also assumes OpenRC. Systemd has much different settings. This is because the Void Linux Guide it is based off is non-systemd native.

This guide is best used when you have already experience installing Gentoo. It is a quick accumulation of how I do my install. Sure I have comments but this is NOT the handbook. Please read that first.

<<Partioning :Step 1>>

lets clean the target disk up, I assume your target disk is /dev/sda. Please check by running lsblk.

<wipefs -a /dev/sda>

<cfdisk /dev/sda>

#select the “dos” partioning table , primary and then bootable. Hit write

<yes>

#then quit out of it. Now we have created our partition. Now lets encrypt the disk

<cryptsetup luksFormat –type luks2 /dev/sda>

#luks2 is now supported on grub so there is no reason to use luks1. Please enter your password and make sure it is secure and that you remmember it. As far as I understand it you can run this command several times before moving to the next step should you forget your password. I am unaware of password length requirments.

Now we open this partition so we can actually populate it.

<cryptsetup luksOpen /dev/sda1 librevm >

“librevm” can be substituted for whatever name you want if it ends in vm. So dunnyvm works as well. I have not tried capital or special characters I would say don’t do that. Now we finally populate it.

<vgcreate librevm /dev/mapper/librevm>

<lvcreate --name root -L 65G librevm>

<lvcreate --name swap -L 16G librevm>

<lvcreate --name home -l 100%FREE librevm>

now this is my favourite partioning scheme you do what you want. Obviously you need a root partition everything else is optional though I would highly recommend swap. Next, we format the partitions we made.

<mkfs.ext4 -L root /dev/librevm/root>
<mkfs.ext4 -L home /dev/librevm/home>

<mkswap /dev/librevm/swap>

#I like this file-system (ext4) just because its built into the kernel by default. Xfs does have known issues with grub so if you are unsure just use ext4. Now we mount the partitions.

<mount –-mkdir /dev/librevm/root /mnt/gentoo>

<mount –-mkdir /dev/librevm/home /mnt/gentoo/home>

<swapon /dev/librevm/swap>

<cd /mnt/gentoo>

#If you are on a gentoo live USB, the directory /mnt/gentoo will already exist but /mnt/gentoo/home does not exist.
#Apparently you can choose whatever mount point you want, so /mnt/gnu is fine as long as you are consistent. So /mnt/gnu and /mnt/gnu/home will work, /mnt/gnu/harry will cause complication when we create the user. This guide will assume /mnt/gentoo = root. This guide assumes that you are using a GUI live environment. If so, then you can use the gui clock settings to set the clock. If not or if you prefer a terminal this next step is for you.

<date MMDDhhmmYYYY>

#for example it is 1am on the 4th of May (May the 4th be with you:) in 2026. So I will say

//example <date 050401002026>

//

Next we grap a stage 3 file. Head over to here.
https://www.gentoo.org/downloads/amd64/#stages

//example <<wget https://distfiles.gentoo.org/releases/amd64/autobuilds/20260503T164604Z/stage3-amd64-desktop-openrc-20260503T164604Z.tar.xx >>

//

Now the handbook goes through a proccess of validating the checksum and all that. This is best practice however I just don’t do it. If you want to please head over there.
https://wiki.gentoo.org/wiki/Handbook:AMD64/Installation/Stage

#Now we extract this file we downloaded.

<tar xpvf stage3-*.tar.xz –xattrs-include=’*.*’ --numeric-owner -C /mnt/gentoo>

#This will take a minute or two.

#Some live GUIs such as the Trisquel live USB have vim included these are better. I will be assuming vim as your text editor because it should be:).

<vim /mnt/gentoo/etc/portage/make.conf>

#This file is highly personalized. It controls every package that will be emerged by Portage. It is essential to get right. So I will instead what I have in mine.

#Under common flags I added -mtune=native -mtune=native this is for perfomance.

RUSTFLAGS=”${RUSTFLAGS} -C target-cpu=native

#Same thing except for Rust

FEATURES=”${FEATURES} parallel-install parallel-fetch candy”

#Lets you install packages in parallel and fetch the source code for multiple packages in parallel. And candy, well try it out and see :)

EMERGE_DEFAULT_OPTS=”-j2 -l3”

#Lets several packages be emerged in parallel. Now I have a Core2Duo which has two threads. If you have a more capable CPU you may have more threads. Run top to find out how many you have. The rule is you need 2G per thread. I have 8GB so I can use 3 fine. So RAM being sufficient,

-j($NPROC) -l($NPROC+1)

#Specific to legacy bios. Again a warning that this guide will not boot at all for UEFI systems. It MUST be legacy bios.

GRUB_PLATFORMS=”pc”

#This will ensure you only have to emerge the bloated grub once.

VIDEO_CARDS=”i965 intel”

#Please check with your hardware

ACCEPT_LICENSE=”-* @FREE”

This is how we block out all that nasty non-free software. This is very, very important. However on Gentoo it is not sufficient. Programs such as Firefox still seep through. At some point I hope to import this into /etc/portage/package.mask properly.
https://libreplanet.org/wiki/List_of_software_that_does_not_respect_the_Free_System_Distribution_Guidelines

GENTOO_MIRRORS=”https://gentoo.osusl.org”

#obviously pick your own mirrors but I have found this one in particular to be really, really, really fast and reliable.
MAKEOPTS=”-j2 -l3”

#Same deal as EMERGE_DEFAULT_OPTS but for CPU useage.

#Now we enter the chroot

cp –-dereference /etc/resolv.conf /mnt/gentoo/etc/

#If you are on gentoo live CD then you can skip the next step with arch-chroot.

<<The next step >>
mount --types proc /proc /mnt/gentoo/proc
mount --rbind /sys /mnt/gentoo/sys
mount --make-rslave /mnt/gentoo/sys
mount --rbind /dev /mnt/gentoo/dev
mount --make-rslave /mnt/gentoo/dev
mount --bind /run /mnt/gentoo/run
mount --make-slave /mnt/gentoo/run
test -L /dev/shm && rm /dev/shm && mkdir /dev/shm
mount --types tmpfs --options nosuid,nodev,noexec shm /dev/shm
chmod 1777 /dev/shm /run/shm

<<Continuing for all>>

```
<chroot /mnt/gentoo /bin/bash
source /etc/profile
export PS1="(chroot) ${PS1}">

<emerge-webrsync>

#We now grab the latest ebuilds from the mirrors.

#If you want binary packages do the next step
```

<<Next step>>

<getuto>

#Next discover what USE flags are set in your profile.

<emerge --info | grep ^USE>

I copy this over to my make.conf but its up to you. Set preferred USE Flags globally with

//example

<USE=”eme-free -telemetry -proprietary-codecs -mp3 -mp4 -mpeg”>

//

The USE Flags shown above are important for your software freedom. EME-FREE eliminates DRM (Digital Rights Managment) capacity from firefox-based browsers (such as librewolf) and firefox based mail clients such as Icedove. It is important to eliminate support for mp3, mp4 and mpeg as these use proprietary patent ridden codecs (h264/h265) that will bypass -proprietary-codecs. The other two are obvious.

<emerge --ask --oneshot app-portage/cpuid2cpuflags &&

echo "*/* $(cpuid2cpuflags)" > /etc/portage/package.use/00cpu-flags>

#This optimizes some packages for you CPU better

<emerge --ask --verbose --update --deep --changed-use @world>

OR

<emerge --ask --verbose --update --deep --newuse --getbinpkg @world>

#--getbinpkg (-g) installs binary packages where possible and not conflicting with your USE flags.

<emerge -av —-depclean>

#This is the Gentoo cleanup command.

< ln -sf ../usr/share/zoneinfo/Europe/Brussels /etc/localtime>

#I assume you know what this does.

< vim /etc/locale.gen >

#uncomment your language of choice. If you are reading this presumably en_US you can search in vim with :/ .

<locale-gen>

<env-update && source /etc/profile && export PS1="(chroot) ${PS1}">

#Now we have set the timezone and the language. And we proceed with Muh Freedoms.

<echo “sys-kernel/linux-firmware -initramfs -redistributable” >> /etc/portage/package.use/linux-firmware>

#You can set the name to anything for example /etc/portage/package.use/johnny-gnu-larry-loves-libre-software

I don’t think dots work, underscores do. By doing this we deblob linux-firmware from the backdoored firmware. As a side affect this will not work on a ton of modern hardware. I have heard that the T580 works once the wifi card is replaced (can only be done after libreboot (libreboot on the T580 contains binary blobs such as the Intel FSP it is not fully free) -initramfs is needed to be set for some reason when -redistributable happens.

<emerge -av linux-firmware sof-firmware>

#sof-firmware is for audio if you don’t care about audio then no need for it.

<echo “sys-kernel/installkernel grub dracut” >> /etc/portage/package.use/installkernel>

#It is important to use Grub here because it will work with Canoeboot (or the non-fully-free libreboot) to unlock your disk.

<mkdir -p /etc/dracut.conf.d>

<blkid /dev/librevm/root or is it /dev/mapper/librevm-root>

#Copy this output to a clipboard or take a photo of it if using a tty install

echo ‘kernel_cmdline=" root=UUID=2122cd72-94d7-4dcc-821e-3705926deecc "’ >> /etc/dracut.conf.d/00-installkernel.conf

#Replace the UUID here with the UUID you copied from the previous step.

<emerge -gav installkernel>

#Using -g to grab binary packages will stop Portage from pulling the whole goshdang Ruby tree.

#Now we configure our keyfile so that canoeboot can unlock our disk.

<vim /etc/default/grub>

#make sure that

<GRUB_ENABLE_CRYPTODISK=y> is in this file or you cannot run grub-install /dev/sda later on.

#Next we discover the UUID we need.

<blkid -o value -s UUID /dev/sda1

#Make sure to replace this with your target disk.

#Here is my lsblk to make it more clear what disk this is. Setting the wrong UUID will result in failure to boot. Lets call the UUID $UUID.

```
 lsblk
[2:43:49]
NAME                MAJ:MIN RM   SIZE RO TYPE  MOUNTPOINTS
sda                   8:0    0 238.5G  0 disk
└─sda1 <--(THIS ONE)              8:1    0 238.5G  0 part
  └─librevm         252:0    0 238.5G  0 crypt
    ├─librevm-root  252:1    0    65G  0 lvm   /
    ├─librevm-swap  252:2    0    16G  0 lvm   [SWAP]
    └─librevm-home  252:3    0 157.5G  0 lvm   /home
```

#Now enter the UUID as such and exactly as such

<vim /etc/default/grub>

<<<GRUB_CMDLINE_LINUX_DEFAULT=”root=/dev/mapper/librevm-root rd.luks.uuid=$UUID rd.lvm.vg=librevm”>>>

<dd bs=1 count=64 if=/dev/urandom of=/boot/volume.key>

#Here I like to use the upstream linux-libre kernel however this is not what Gentoo suggests since you cannot update it with Portage. However this is much faster than deblobbing gentoo-sources. This step is important to ensure that your computer is secure. The stock linux kernel includes binary blobs and firmware loading hooks for dangerous backdoored firmware. Linux-libre deblobs this. If you want to manually deblob a kernel go ahead with that now. https://wiki.gentoo.org/wiki/Kernel/Deblobbing

Due to the lack of an ebuild for linux-libre-dist we will have to manully configure this kernel although it may be possible to use a binary. I have not tried this.

<cd /usr/src>

<wget https://linux-libre.fsfla.org/pub/linux-libre/releases/LATEST-7.0.N/>

#This is the latest kernel as I am writing this obviously check here https://linux-libre.fsfla.org/pub/linux-libre/releases/.

<tar -xpvf <<file-name>> >

<eselect kernel set 1>

#This is an important step since Portage checks kernel sources against the set kernel.

<cd linux >

#The full directory name is /usr/src/linux

#We now must configure our kernel to be able to run dm-crypt, the primary application on cryptsetup which we will use to unlock our encryption. First we need to emerge these packages.Lvm2 is important since we are using lvm as our volume it will be needed to deliver root at boot.

<emerge -av cryptsetup lvm2>

<cryptsetup luksAddKey /dev/sda1 /boot/volume.key>

<chmod 000 /boot/volume.key>

<chmod -R g-rwx,o-rwx /boot>

<echo “librevm /dev/sda1 /boot/volume.key luks” >> /etc/crypttab>

<echo ‘install_items+=” /boot/volume.key /etc/crypttab “’ >> /etc/dracut.conf.d/10-crypt.conf>

#ENSURE ENSURE ENSURE that lvm2 has the “lvm” USE flag enabled or you will be dropped into emergency mode when you reboot.

#We have now configured our keyfile so that canoeboot can find our disk and unlock it.

#These will deliver missing kernel settings that you can use. But I will review them here. Some devices such as my thinkpad x200 have actual documentation on what settings are needed. However this is not there for all devices. This will NOT cover that since that scope is massive. You can search on menuconfig the same way as in vim with “/”. You can type “h” to learn more about the highlighted item. Good luck

//Config settings

First under General setup → Local-version -append-to-kernel-release choose your own custom cool name.

Next, under device-drivers → graphics-support → framebuffer-devices set to =y then select your choice of framebuffer device. Then go set bootup-logo=y and now when you reboot, if it works,you get a Tux penguin per CPU thread :)

Now to serious stuff. Here are the config settings you need. Use the search utility to find them and set them to =y (using “y” if you set one by accident hit “n” or the space bar.Cycle through options in the menu with vim keys and select with the number note that 3 will not be avaliable until the number “3” is on screen)

CONFIG_MODULES

CONFIG_MD

CONFIG_BLK_DEV_DM

CONFIG_DM_CRYPT

CONFIG_CRYPTO

CONFIG_CRYPTO_AES

CONFIG_CRYPTO_XTS

CONFIG_CRYPTO_SHA256

CONFIG_CRYPT_USER_API_HASH

CONFIG_CRYPTO_USER_API_SKCIPHER

CONFIG_BLK_DEV_INITRD

CONFIG_BLK_DEV

CONFIG_BLK_DEV_LOOP

CONFIG_FUSE_FS

CONFIG_CRYPTO

CONFIG_CRYPTO_RMD160

CONFIG_CRYPTO_SHA512

CONFIG_CRYPTO_WP512

CONFIG_CRYPTO_LRW

CONFIG_CRYPTO_SERPENT

CONFIG_CRYPTO_TWOFISH

//settings for lvm

Device-Drivers →
  Multiple devices driver support (RAID and LVM) =y
    Device mapper support =y
      Crypt target support =y
       Snapshot target =y
       Mirror target =y
       Multipath target =y
          I/O Path Selector based on the number of in-flight I/Os=y
          I/O Path Selector based on the service time =y

//for added perfomance

 CONFIG_X86_NATIVE_CPU=y

Once these and your other hardware settings are set its time to compile the kernel. But first we need some dependencies.

<echo “sys-libs/binutils-libs 64-bit-bfd” >> /etc/portage/package.use/binutils-libs>

<emerge -av sys-devel/bc sys-libs/binutils-libs>

#Non-desktop stage files will need to emerge harfbuzz as well. If you set a different compression such as lz4 you would need to install support for that here (app-arch/lz4). Great now we can compile it make sure you are using all the threads you have RAM for.

<make -j2 && make modules_install && make install>

#Next we have to fill our /etc/fstab. Make sure every partition you made is mounted. Some people like to fill in their fstab themselves. I do understand

why you would want to do this. It is explained in the handbook under Configuring the system.

<emerge -av genfstab>

<genfstab -U / >> /etc/fstab>

<clear && cat /etc/fstab>

#Check the output if it has everything you need it should, if you are a following along, /home, / and swap.
cat /etc/fstab

<fs> <mountpoint> <type> <opts> <dump> <pass>

#LABEL=boot /boot ext4 defaults 1 2
#UUID=58e72203-57d1-4497-81ad-97655bd56494 / xfs defaults 0 1
#LABEL=swap none swap sw 0 0
#/dev/cdrom /mnt/cdrom auto noauto,ro 0 0
# /dev/mapper/librevm-root LABEL=root
UUID=6450978e-1078-485d-80c1-94e7bd4c1ddc / ext4 rw,relatime 0 1

# /dev/mapper/librevm-home LABEL=home
UUID=6d597504-93f3-493d-930f-f52cd522e02c /home ext4 rw,relatime 0 2

# /dev/mapper/librevm-swap
UUID=1a4c661e-19ce-46de-905b-2fddf43dda30 none swap defaults 0 0

Now we make our hostname.

< echo larielfreedom > /etc/hostname >

<vim /etc/conf.d/hostname >

#replace “localhost” with your hostname or your hostname will be hijacked when connecting via NetworkManager.

#The next steps are regarding network tools. I prefer NetworkManager because its easier however it is very bloated.

For dhcpcd it should be installed already

<rc-update add dhcpcd default>

for NetworkManager not so much.

#If you want to use things like nmtui and nmcli >>

<echo “net-misc/networkmanager tools” >>
/etc/portage/package.use/networkmanager>

<emerge -av networkmanager>

#official docs tell you to add networkmanager to your global use flags in make.conf but I have not seen this flag anywhere at all so I am not including it.

#if you want networkmanager you cannot have any other network service. Once you make sure of this.

<rc-update add NetworkManager default>

<passwd>

set the root password

#Next we install system tools. The Gentoo page is just better at this with many options however if you want something quick and functional here you go….

<emerge -av app-admin/sysklogd sys-process/cronie sys-apps/mlocate app-shells/bash-completion net-misc/chrony sys-block/io-scheduler-udev-rules && rc-update add sysklogd default && rc-update add cronie default && rc-update add chronyd default >

#Thanks to the USE flag settings on installkernel and in our make.conf we can just install grub again no need to re-emerge it.

<grub-install /dev/sda>

#make sure /dev/sda or whatever is your target disk

<grub-mkconfig -o /boot/grub/grub.cfg>

<exit>

<umount -R /mnt/gentoo>

<reboot>

GOOD_LUCK!!!!